# OpenCUDA+MPI

## A Framework for Heterogeneous GP-GPU Distributed Computing

Kenny Ballou
College of Engineering – Department of Computer Science
Boise State University
Alark Joshi, Ph.D

**Abstract**

The introduction and rise of General Purpose Graphics Computing has significantly impacted parallel and high-performance computing. Even worse, it has brought about even more challenges when it comes to distributed computing (with GPU's). Current solutions target specifics: specific hardware, specific network topology, a specific level of sameness. What if we could ignore specifics? What if we could write a general algorithm to solve a problem and have a job scheduler do the rest? That is the goal of OpenCUDA+MPI. To achieve this goal, we have written a framework that allows a developer/ data scientist to write a general algorithm/ solution without the overhead of worrying about the specifics of the cluster it will run against. We have found, [nothing yet] and, therefore, can conclude [also nothing].

**Keywords:** Parallel Computing, Distributed Computing, General Purpose Graphics Processing, High-Performance Computing, Scientific Computing

# 1 Introduction

Graphics Processing Units (GPUs) have significantly altered the way high-performance computing tasks can be performed today. A major challenge remains in being able to seamlessly intergrate multiple workstations to further parallelize the computational tasks. Current approaches provide the ability to parallelize tasks, but they are less focused on optimally utillizing the varied capabilities of heterogeneous graphics cards in the cluster of workstations.

## 1.1 Outline

## 1.2 What Others Have Done

## 2 Methodology

Our framework will be developed using a number of tools and will also build upon a few already developed and mature software libraries. Overall, we plan to use the Python programming language, Message Passing Interface (MPI) for cross process communication, and Compute Unified Device Architecture (CUDA) for graphics computing.

### 2.1 Python

We will use Python because of its ease of development and plethora of existing libraries, as we will mention further below.

Python also adds some advantages when it comes to usability when needing more performance. That is, if we discover a need for certain aspects of the project to be tuned and otherwise run faster, we can easily drop into C++ and write sections of the program in a more native language.

### 2.2 MPI

We will use MPI because it has established itself as the de-facto interface for cross process communication [citation needed]. To allow for interfacing with Python, we will use `mpi4py` because of its maturity and implementation completeness.

### 2.3 CUDA

Further, we will be using CUDA because of existing knowledge of the framework and also because it is a well established framework for GPU computing.

## A Reference

[1] Lisandro Dalcin. Mpi for python. http://mpi4py.scipy.org/docs/usrman/index.html, Janurary 2012.

[2] Wes. A comprehensive mpi tutorial reference. http://www.mpitutorial.com/, 2012.

## Glossary

| | |
|---|---|
| **CUDA** | Compute Unified Device Architecture. 3 |
| **GPU** | Graphics Processing Unit. 2, 3 |
| **MPI** | Message Passing Interface. 3 |